Journal of Imaging Science and Technology® 60(1): 010408-1–010408-10, 2016.

End-User Development of Visualizations

Kostas Pantazos† **and Soren Lauesen**
IT University of Copenhagen, Rued Langgaards Vej 7, DK-2300, Copenhagen S., Denmark
E-mail: slauesen@itu.dk

Abstract. *In this article the authors investigated a visualization tool (uVis) for end-user developers, in order to see how end users actually use it. The tool was an early version and the investigation helped the authors to improve it. The investigation showed that users appreciated the simple formula language, the coordinated panels, and the drag-and-drop mechanism. However, the most important thing for them was the immediate response when they changed something, for instance part of a formula. The entire visualization was updated immediately without having to switch from development view to production view.*

With uVis, developers construct a visualization from simple visual components such as boxes, curvePoints, and textboxes. All component properties such as Top and BackColor can be complex formulas similar to spreadsheet formulas. The operands in the formula can address relational data in a database, other visual objects, and dialog data provided by the user. A special Rows property can bind to a database query and make the component replicate itself for each row in the query. In this way, traditional as well as novel visualizations can be constructed.

The most serious usability problems were data binding and not noticing errors (errors were shown in an error list, but not in the formula that had the error). There were many other usability problems. Removing them would speed up learning and make the tool more successful. © *2016 Society for Imaging Science and Technology.*
[DOI: 10.2352/J.ImagingSci.Technol.2016.60.1.010408]

INTRODUCTION

The provision of good data visualizations is challenging. A good understanding of the data is necessary, and several visualizations must be tried and evaluated by domain experts. Good visualizations will allow users to accomplish tasks effectively and efficiently (usually a problem according to Ref. 1). Traditionally, visualization development is a collaboration between domain experts and professional programmers. Both parties spend time and resources to design a good visualization. Usually, there are communication problems in this user-centered design practice:[2] users have the domain knowledge but no programming skills, while programmers do not have the domain expertise.

In the last decade, a new research discipline has emerged, End-User Development (EUD). End-user developers are users who *may have little or no formal training or experience in programming*.[3] The main goal of this discipline is to empower users so they can create, modify, and extend software artifacts, and as a result gain more control over their applications.[4,5] Taking into consideration the predictions from Ref. 6 and observing the increased use of computers, it is important to investigate how end-user developers can enter the visualization area. However, first we have to investigate whether they actually can create visualizations and what is required to make them do it. Investigation of this path is an important factor for the advancement of EUD and information visualization (InfoVis), which has been discussed in prior research (e.g., see Refs. 7–9).

In response, we developed a visualization tool for end-user developers, uVis, and conducted two usability studies with end users. The first study involved nine end-user developers in three-hour sessions. The second study involved two clinicians in two-hour sessions. In both studies, participants were asked to create a predefined custom visualization from scratch.

The results showed that with a modest amount of training, end-user developers can construct visualizations. The most difficult part was data binding (all of them encountered this problem at least once). Future research should investigate novel approaches to improve the understanding of the relation between data and visualization. This study also showed that the most appreciated feature for the participants was to test the application without switching workspace from development view to production view. Furthermore, different strategies observed during the tests and from their rating indicated requirements for a flexible development environment. It seems that one solution does not fit all, and a combination of panels yields a better understanding and better visualizations.

The rest of the article is organized as follows. First, we discuss related work on end-user development and information visualization. Next, we describe uVis and the two studies. The article concludes with a discussion of limitations and directions for future work.

RELATED WORK

As this work concerns end-user development and information visualization, we reviewed the literature in both areas.

End-User Development

The End-User Development (EUD) field is a research discipline that has emerged from research in Human–Computer

† Kostas Pantazos passed away on October 2015. He received his BS in informatics from the University of Athens (2005) and his PhD in software engineering from IT University of Copenhagen (2013). Since then he worked as a postdoc at Copenhagen Business School, and recently as an external lecturer at the IT University of Copenhagen.

Received June 26, 2015; accepted for publication Nov. 24, 2015; published online Jan. 19, 2016. Associate Editor: Song Zhang.

1062-3701/2016/60(1)/010408/10/$25.00

Interaction, Cognitive Science, Requirements Engineering, Software Engineering, Computer-Supported Cooperative Work (CSCW), Artificial Intelligence, Information Systems, and the Psychology of Programming.[10] Lieberman et al.[4] defines EUD as "a set of methods, techniques, and tools that allow users of software systems, who are acting as non-professional software developers, to at some point create, modify and extend a software artifact." Consequently, end-user developers are not professional programmers, but users who *may have little or no formal training or experience in programming*.[3] The main purpose of EUD is thus to empower users so that they can create, modify, and extend software artifacts, and as a result gain more control over their applications. Examples of end-user developers are system administrators, interaction designers, teachers, accountants, health care workers, and managers.

Rode et al.[11] investigated EUD of web applications. Their study showed that web development tools focus on supporting developers with a wide range of functionalities. Less attention is paid to ease of use. This finding was also observed in another study that surveyed EUD and visualization tools.[12]

Lieberman et al.[4] define two types of end-user activities: parameterization and customization (activities that allow end users to parameterize or customize their applications using existing presentations) and program creation and modification (activities that allow end users to create or modify software artifacts). In order to support these types of activities, the system should be flexible and expressive enough to set parameters, compose objects, etc.[4] Simple changes are not difficult, but things become more complicated as ambitions increase. MacLean et al.[13] suggest a "gentle slope" to reduce the level of complexity and support development on different levels.

Information Visualization

The Information Visualization field (InfoVis) has enabled the development of visualization systems that enhance human cognitive processes by visually presenting data.[14] Considering the variety of data and user tasks, it is obvious that new visualizations have to be developed all the time. Several InfoVis toolkits and tools have been developed to assist users and improve visualization development.[15–24] Most of them have ignored empirical evaluation with end-user developers (e.g., Refs 16–18), focusing primarily on case studies and novel visualizations developed by programmers.

Isenberg et al.[25] presented an assessment of the evaluation practices in articles from the IEEE visualization conferences, and concluded that the work on visualization often does not involve any participants at all.[25] Another study[12] surveyed end-user developers of InfoVis, reviewing existing InfoVis tools and how they constructed predefined and custom visualizations. The survey showed that end-user developers need better tools to create and modify custom visualizations.

User-centered Design

To facilitate the visualization development process and ensure that visualizations provide adequate information, several InfoVis applications (e.g., Refs. 26–29) have been developed with user-centered design, where users participated during the entire development process. Norman[30] and Nielsen[31] describe user-centered design as the early and continuous involvement of end users in the design and development process.

Considerable work has been conducted to define activities in the user-centered model for design and implementation of InfoVis tools.[2,27–29] For example, Robinson et al.[28] describe a six-stage user-centered design process where users are involved and provide input in each stage.

Using this model,[28] Roth et al.[29] present a modified user-centered design approach that starts with prototyping, followed by interaction and usability studies, work domain analysis, conceptual development, and implementation, and ends with debugging.

Although the user-centered model helps in producing better visualizations, it is hard to bridge the gap of knowledge between end users and programmers. This gap can create challenges such as the following: programmers should understand end-user needs, end users should gain some knowledge of InfoVis, end users should be devoted to and actively participate in the process, etc. The study by Koh et al.[2] reports on such challenges.

This article focuses on the development and modification of visualizations by end-user developers, i.e., users with limited programming skills. The purpose was to investigate how they create visualizations and to elicit requirements for improved tools.

uVIS

Below we present a visualization tool for end-user developers. It consists of the following.

- uVis Studio: The Studio has several panels similar to many other development environments. Many of the panels are coordinated to show the same data in different ways. The developer can drag and drop visual objects (controls) to the Forms in the central design panel and define the properties of the visual objects by means of the Property Grid (lower right).
- The Formula Language: In contrast to most other development tools, a property can be a complex formula. The operands in the formula can address relational data from a database, other visual objects, and dialog data provided by the user.
- Rows property: Each visual object has a Rows property that can make the object generate several instances of itself. The instances share the same formulas, but the results of the formulas may vary from instance to instance. In this way, a single visual object can generate, for instance, all of the nodes in a line graph or all of the pie slices in a pie diagram. The Rows formula will typically be a database query expressed in a very

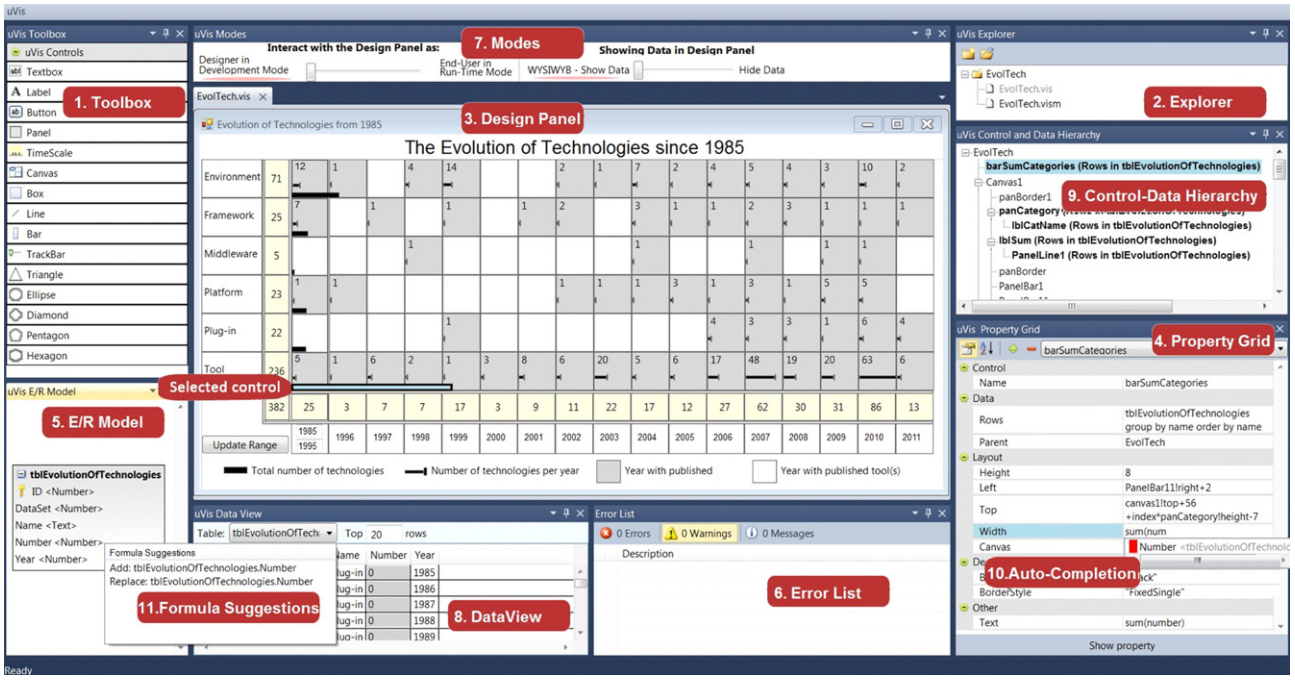


Figure 1. A custom visualization being developed in uVis Studio. The Design Panel shows the evolution of technologies since 1985. The yellow column and row show the total number of technologies per category and year. The wide black bars show the total number of technologies. Each box uses color coding to show whether there have been publications or not. The narrow bar inside a box shows the number of technologies per year. The first column aggregates the number of technologies for a range of years, which is defined in the two text boxes. End users can change the years and the visualization updates. The developer has selected a control for modification (the dark blue border in the Design Panel). It is highlighted in the Control-Data Hierarchy and its property formulas are shown in the Property Grid, where the developer can edit them.

compact way. It will cause the visual object to generate an instance of itself for each row in the query. Moreover, the Rows formula can address visual data and dialog data, for instance in the Where clause.

Figure 1 is a screen shot of uVis Studio. The central Design Panel shows a custom visualization created by an end-user developer (it visualizes the evolution of technologies).

A uVis application is a folder that contains a vis-file for each Form in the application and a vism-file (map file) that specifies the connection to a database and the initial Form to open (see Figure 2). This version of uVis was designed after feedback from tests with programmers using the first version.[15]

The uVis Formula Language

Developers construct visualizations by placing visual objects on Forms and specifying uVis formulas for their properties, as explained above. Figure 3 shows two steps in creating a simple bar chart. In (a), the developer has put a simple box control on the Form and made the background blue, using the BackColor property. In (b), the developer has specified the Rows property to be a simple database query that retrieves all rows of a table. As a result, the box will generate an instance of itself for each row. The developer has also specified formulas for the size and position properties. They compute their values based on data in the row and the row number (index). As result, we see a bar graph. The developer might also make the BackColor a formula, and as a result each bar could have a data-dependent color.

uVis provides simple controls, e.g., Label, Textbox, Box, Glyph (a control with data-dependent shape), and GraphPoint. There are also a few advanced ones, e.g., a TimeScale that can show several periods of time with different zooms and align other controls according to time.

Controls have three kinds of properties, as follows.

- *Value properties:* All controls have common properties such as *Top, Left, Width, Height, BackColor*, and *Rows*. Each type of control can, in addition, have its own special properties such as *Text, Radius*, or *Shape*.
- *Designer properties:* Any control can have additional properties with names defined by the developer. He/she may, for instance, write a complex formula in a designer property and let other properties or controls refer to it.
- *Event Properties:* They do not have a formula that computes a value, but one or more statements that are executed when the event happens. As an example, the click event handler for a button may assign values to properties (overriding the formulas), assign values to database fields, call *Refresh()* to update the screen, or call OpenForm(. . .) to show one more Form on the screen. At that time there were no conditional statements or loops.

uVis Studio

The uVis Studio had nine panels at that time, as shown in Fig. 1.

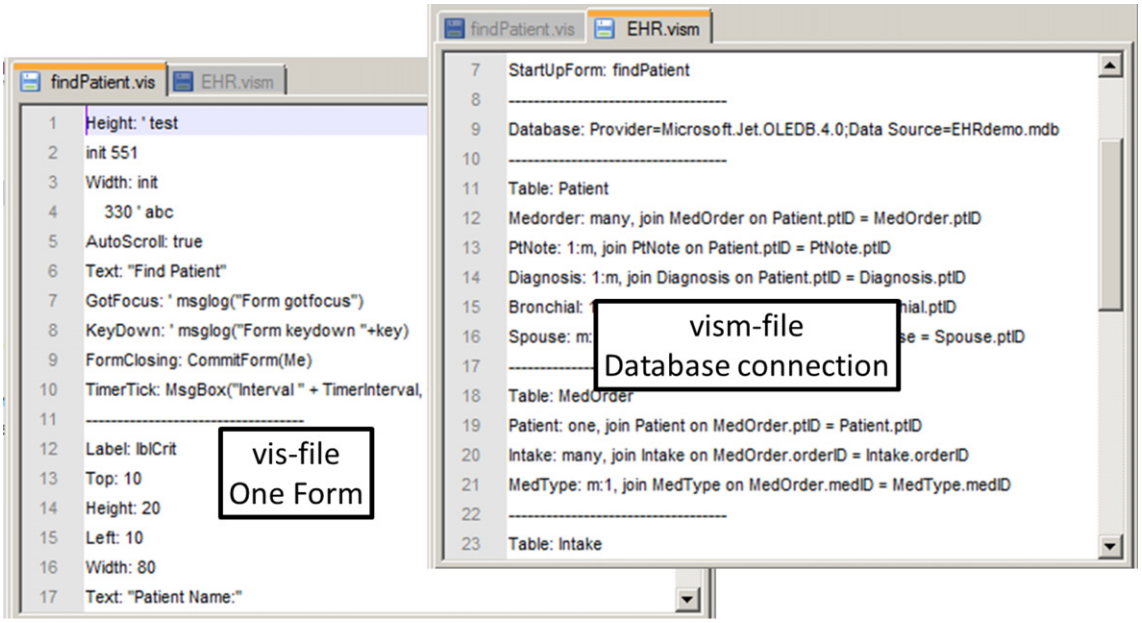
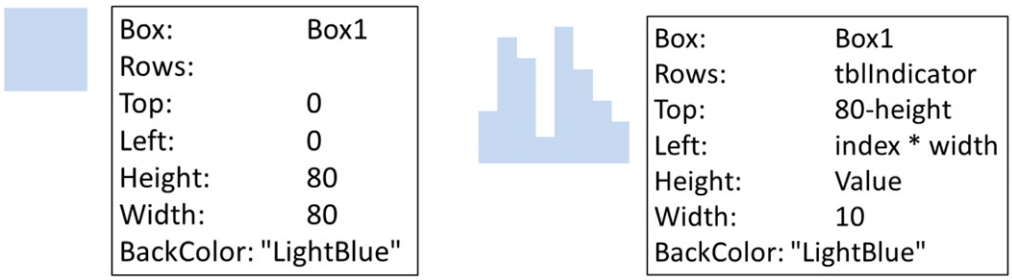


Figure 2. A vis-file defines a Form. A vism-file defines a connection to a database.



(a) (b)

Figure 3. Two steps when creating a bar chart. In (a), the developer has put a simple box on the Form and made the background light blue, using the BackColor property. In (b), the developer has specified the Rows property to be a simple database query that retrieves all rows of table *tblIndicator*. As a result, the box will generate an instance of itself for each row. The developer has specified size and position properties that depend on the data. The height of the box is *Value*, a field in the data row, and the Left property is *index*width*, where *index* is the row number. As result, we see a bar graph.

(1) *The Toolbox* contains the control types that uVis supports, such as label, button, timescale, etc. The developer drags and drops a control into the Design Panel and Studio sets the default property values.

(2) *The Explorer* shows visualization files (vis-files) and data mapping files (vism-files). When the developer clicks on a vism-file, uVis connects to the database and opens the initial Form in the design panel. When he/she clicks on a vis-file, the Form opens in the design panel.

(3) *The Design Panel* shows a Form with one or more visualizations. The developer drags and drops controls from Toolbox, moves and resizes them. However, it was not possible to move a control when it had a formula for the position (*left* or *top*). Whenever the developer selects a control, it becomes highlighted with a blue frame and the corresponding formulas appear in the Property Grid. The control is also selected in the Control-Data Hierarchy, and the DataView shows its row data. The Design Panel is "live." It supports direct manipulation and provides continuous feedback during development. When the developer has changed a formula, the Design Panel updates immediately. The update reuses database data that have been retrieved already. This is called What You Bind Is What You See (WYBIWYS).

(4) *The Property Grid* shows the properties for the selected control. A row in the Property Grid shows the property name and the formula. Properties can be sorted alphabetically or shown in groups to help the developer to find them faster. A change in the Property Grid is immediately reflected in the Design Panel. Moreover, a change in the Design Panel (e.g., dragging a control) automatically updates the Property Grid. Writing formulas can be challenging, as developers must remember the syntax, and it is common to misspell words. To help them, the Property Grid has Auto-Completion which suggests what can follow. The suggestions can be available variables and language

functions, and also suggestions for tables, table fields, and relationships in the database.

(5) *The E/R Model* uses the vism-file to extract tables, fields, and relationships from the database. The E/R Model shows the database as an Entity Relationship diagram (E/R).[32] As the database may contain many tables, developers can expand or collapse table fields and detach the E/R Model panel from the Studio to enlarge it and get a better overview. The E/R Model has a feature called Formula Suggestions. This feature helps users to specify the row formulas.

(6) *The Error List* shows a list of errors detected in the formulas. The developer double clicks the error, and the wrong formula in the Property Grid appears and is colored in red. Once the formula is changed, the error list updates. In spite of errors, the visualization shows all of the time. When a formula has an error, uVis uses a default value.

(7) *The Modes* panel is positioned above the Design Panel. It shows the Interaction-Mode (interaction with the Design Panel as an end user or as a developer) and the Data-Mode (whether the Design Panel shows data or formulas).

(8) *The DataView* panel shows data from the database. This panel is coordinated with the E/R Model. The developer clicks a table in the E/R Model, and the rows are shown in the DataView panel. The developer may also click a field, and the column is brought to focus and highlighted.

(9) *The Control-Data Hierarchy* shows the data hierarchy in the form, displayed in a tree format. If control A is bound to a database table and control B is bound to a table that has a one-to-many relation to A, we say that B is a data child of A. Each control is represented by a node and the data children as sub-nodes. The text style is bold if the control or its Parent control is bound to data.

The "liveness" factor

uVis visualizations are constructed with a Drag–Drop–Set–View–Interact approach. Developers drag and drop controls, set control properties using formulas, immediately see the new visualization, and immediately interact with the visualization as end users. Tanimoto[33] defined a taxonomy of development environments based on how fast the developer gets feedback, known as the "liveness" taxonomy. This taxonomy has four levels of liveness. uVis Studio is highly "live" and falls somewhere between levels three and four.

Other Approaches

uVis resembles and has been inspired by existing visualization tools. For example, the formula principle is inspired by Excel and adjusted in order to access database items from the formula. uVis utilizes the same principle as Protovis: it uses the principle of building blocks, but does not require knowledge of three languages (i.e., html, svg, and css). In addition, uVis has an integrated development environment (IDE) that Protovis did not have at that time. The IDE is somewhat similar to the IDE of Tableau, Spotfire, and MS Visual Studio. Existing tools mainly provide visualizations that cannot be adjusted at the building block level, while uVis formulas work on that level.

EVALUATION

We conducted two usability studies of uVis with end-user developers as test subjects.

Usability Study 1

Participants

Participants were invited by email from our network at the university. We specified the required skills and IT knowledge. Sixteen participants responded. However, only nine participants met the requirements and participated in the study (three females and six males, between 20 and 35 years old). Three of them were master students, four were PhD students, and two were employees. None of them had used uVis, but all of them had used MS Excel to create a standard visualization (e.g., a bar chart). None of them had experience with designing or developing visualizations.

Activities

Each usability test had one participant, took three hours, and consisted of two activities. The tests were conducted at our university, using a laptop connected to an external monitor to give an introduction to uVis. We explained the uVis formula language and the Studio through a video. During the construction of the visualizations, we assisted the participants whenever there were misinterpretations, confusions, or malfunctions of the system. Whenever we had to assist the participant, we recorded it as a usability problem.

Activity 1—Introduction to uVis. This lasted around one and a half hours. In the first 20 min, we asked some background questions, briefly explained the uVis formula language through a reference card (cheat sheet), and showed uVis Studio on paper. Next, the participants followed a video (played on the laptop screen) and replicated the steps in uVis Studio. The video showed step by step how to create a bar chart (Figure 4). The video was 10 min long. It had no audio, but several call-outs to explain how to use the uVis formula and the Studio. The participant was informed from the beginning that he/she was allowed to stop the video, and play it backwards and forwards. At the end of activity 1, the participant was asked to rate his/her experience with uVis using a five-point Likert scale: 1—strong disagreement . . . 5—strong agreement.

Activity 2—Construct a Process-Completion Diagram. We presented a special visualization, a Process-Completion Diagram (PCD),[34] using a scenario from the medical domain, and explained how the event-log data were visualized. The participant was asked to create such a PCD (Figure 5). We encouraged the participant to do it in a think-aloud manner. The steps involved are representative ones in visualization development (e.g., drag and drop controls, refer to properties, bind controls to data, map a field to a property, view results, etc.). At the end, the participant answered the same questionnaire as in activity 1.

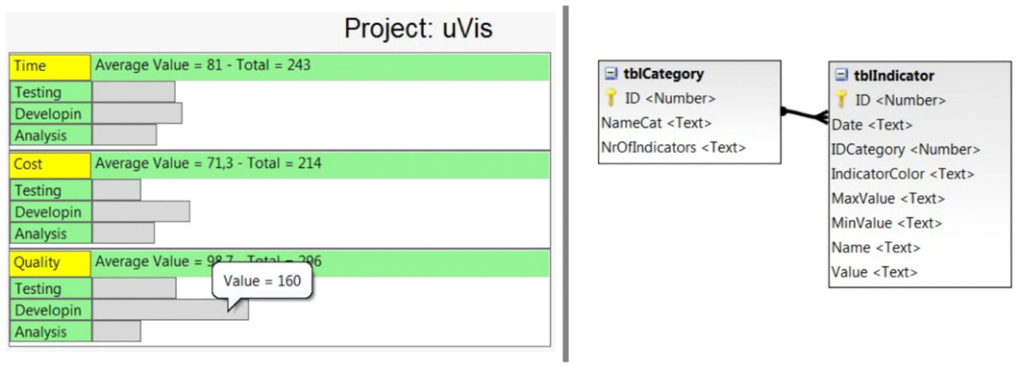


Figure 4. A bar chart and the E/R Data Model behind it.

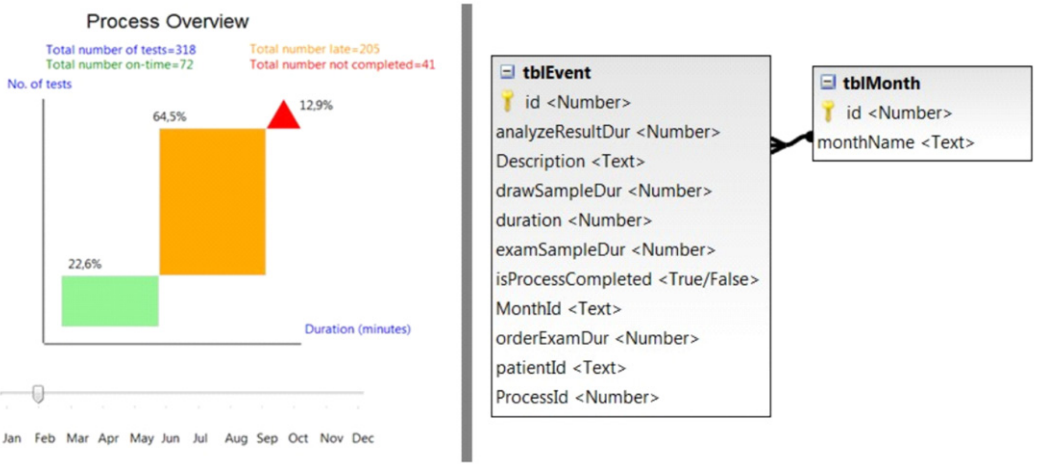


Figure 5. A Process-Completion Diagram (PCD) used in this study and the E/R Data Model behind it.

Data Collection

We collected qualitative and quantitative data. More specifically, we obtained data from interviews, observations, the think-aloud approach, and questionnaires. During each usability test, we kept notes and voice-screen records for later analysis.

Each recorded problem was classified using the problem classification in Ref. 35. For instance, in cases where the system failed to perform correctly, we classified the problem as a *bug*. In cases where participants could not complete a step on their own, it was recorded as a *task failure*. When participants complained about something, we recorded it as a *cumbersome* problem.

Results

In this study, we identified 44 distinct usability problems based on 154 problem observations: 12 task failures, two cumbersome, two medium, six minor, nine bugs, and 13 cases of missing functionality. Participants faced several problems, and all of them failed at least twice to accomplish the task on their own. Each problem occurred more than once. For instance, 47 task failures were observed, but they comprised only 12 distinct problems. Similarly, 29 cases of missing functionality were observed, but comprised only 13 distinct functionalities. A missing functionality corresponds to something that the system did not have, but the user expected. Figure 6 shows all of the problems recorded in this study.

Problem type	P1	P2	P3	P4	P5	P6	P7	P8	P9	Observed	Problems
Task failure	2	11	3	3	8	5	3	9	3	47	12
Cumbersome	2	2	2	2	2	2	2	2	1	17	2
Medium	0	2	0	0	1	0	0	1	0	4	2
Minor	0	2	2	2	3	2	2	0	0	13	6
Bug	5	5	5	6	5	5	5	4	4	44	9
Missing Functionality	2	4	3	6	2	3	2	4	3	29	13
Total										154	44

Figure 6. Number of problem observations and number of distinct usability problems for the first usability study. Each distinct problem was observed several times.

Despite their initial indication of their computer skills, we classified two participants as users with too low computer skills to be end-user developers. However, because we noticed it late in the process (while doing the study), we decided to run and record the tests anyway. Their problems are included in the statistics.

Usability Problems

Below we comment on some of the observations.

(1) Creating several instances of a control by binding it to rows of a table was understandable to most participants. However, it was hard in the case where they had to use a Group By. During the study most of them reasoned correctly, but they were not sure how to express it in the formula language. This indicates the need for better mechanisms for data binding. However, after the first attempt, participants (apart from #2 and #8) could proceed on their own, indicating that learning can be overcome.
(2) Apart from participant 8, all participants could easily refer to properties—also in other components. With the help of Auto-Completion and a bit of experimentation, they mastered it. In the evaluation they found it easier to refer to properties than to database data.
(3) The uVis formula language has several keywords and operators. Some of them were familiar and quickly understood, others made participants skeptical or confused. For example, two participants could not distinguish the difference between the *dot* (used for referring to database fields) and the *bang* (used to refer to components and properties). Seven of them could guess that the left-join operator -< generated rows by joining two tables.
(4) We noticed from the think-aloud protocol that participants found the Design Panel useful. Further, interacting with visual objects was easy, and the WYBIWYS feature continuously helped them to understand the formulas and their actions. Participants were using Auto-Completion to avoid misspellings and also to comprehend what they were typing. We observed this when they stopped typing, either because they misspelled something or because they wrote a wrong formula.
(5) uVis Studio provides several coordinated panels. Changes in the Property Grid were immediately reflected in the Design Panel, the Control-Data Hierarchy, the Error List, and the DataView. This allowed participants to view changes from different perspectives and improve their understanding of the formula language.
(6) All participants had difficulties in identifying the errors.
(7) Participants found it useful to see the E/R Model and the DataView panel, because they often referred to them before writing a formula. We observed several strategies for using these panels. Some of the participants used mainly the Control-Data Hierarchy to navigate through controls and view their properties in the Property Grid. Others used the Design Panel. These strategies confirm the need for a flexible set of panels.

Questionnaires

The questionnaires captured the subjective ratings of each participant about general aspects of uVis, the formula language, and features and panels of the development environment. These ratings comply with our observations and provide useful insights about end-user developers' requirements. The ratings show that participants found the development environment more interesting than the formulas. They appreciated features/panels such as Auto-Completion, Design Panel, Control-Data Hierarchy, Modes. Nevertheless, several aspects in the visualization development were confusing and less appreciated than others; for example, the formula language, the error notification mechanism, and data binding.

Debriefing

Participants' comments varied from positive to negative. One participant said "It will be really easy to create visualizations with this tool, but users need some IT and database knowledge." This once more points out the difficulty of data binding. Another one mentioned "I think the Studio helps you, especially the Design Panel." The right balance between the formula language and the development environment was pointed out by one of the participants, who said "To find the right balance of the formula language is probably a delicate matter, but using the software for the first time was a bit too technical." Another participant commented "It reminds me of Gemini, a tool I used in the bank in 2008. I liked the panels, the drag and drop, and viewing the changes in the Design Panel." The following comment reports on data binding and error notification: "Binding controls to data is not easy, especially when you have to use a Group By. Also, there are errors that I several times did not notice."

Usability Study 2

One of the goals of uVis is to be applicable in hospitals. Therefore, we conducted a usability test with two clinicians who had computer skills that correspond to being an end-user developer. However, clinicians are very busy all of the time, so we could not repeat the approach from the first study. Therefore we adjusted the procedure to save an hour, but retained the core principles for using uVis.

Participants

The first clinician was Søren Lippert, a former Senior Surgeon, 62 years old. He was a part-time PhD student and member of the uVis project. He had worked with Health Informatics since 1994. He provided the medical expertise in the project. He had started using a very primitive version of uVis a month before, but only around one day a week. He had been using a simple text editor to write formulas since uVis Studio did not exist yet. He had designed visualizations using pencil and paper, but had never programmed one.

The second clinician had no prior experience with uVis. He was a Senior Surgeon, 57 years old, started using IT in 1985, and had done a bit of programming using Visual Basic. He does not program often and mostly uses MS Access.

Activities

This study was planned to run in two hours. It consisted of two parts: introduction and constructing a LifeLine visualization of a health record. The first usability test was conducted at the university. The second usability test was conducted at the clinician's office.

Activity 1—Introduction to uVis. This activity lasted 30 min, where we explained the uVis formula language and

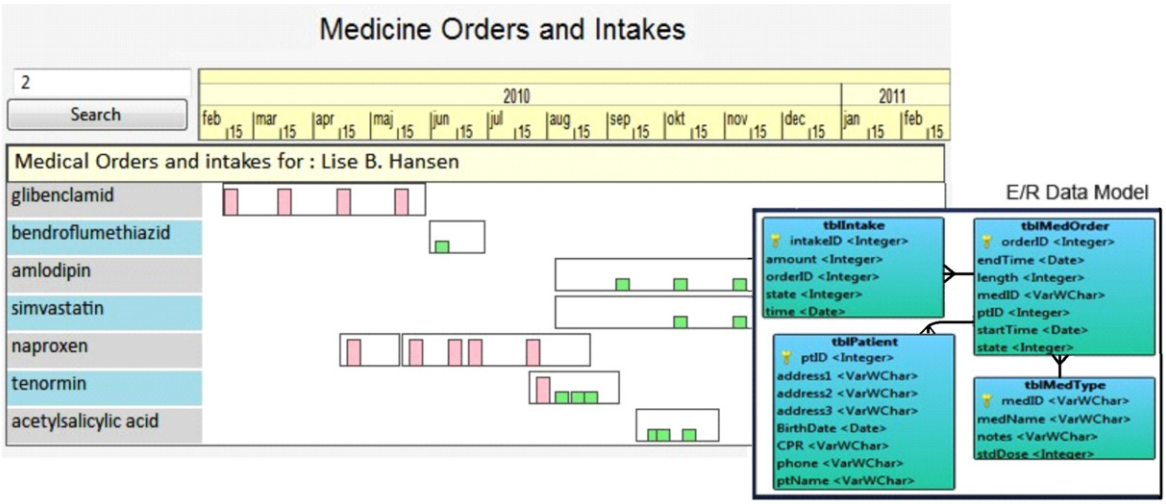


Figure 7. A LifeLine health record and the E/R Data Model behind it.

the Studio. We explained how visualizations are created, what the uVis formulas are, how to refer to control properties, to data, etc. Next, we gave the clinician a list of what he was supposed to do. We showed the clinician the reference card for the formula language. The clinician was asked to complete this activity because it would familiarize him with the formulas and the Studio. At the end of this part, the clinician was asked the same questions as in the first usability study.

Activity 2—Construct a LifeLine. We showed the clinician an already implemented version of a custom visualization of a LifeLine health record, which used data from four tables. Figure 7 shows the visualization and the E/R Data Model. We explained some advanced concepts (i.e., use -< to bind data), and asked the clinician to create a similar LifeLine and think aloud during the process. At the end, the clinician was asked to reply to the questionnaire again as in activity 1. This activity was planned to last one and a half hours.

Data Collection

The same procedure was followed as described in the first study.

Results

We recorded 29 distinct usability problems based on 53 problem observations: seven task failures, three cumbersome, three medium, four minor, four bugs, and eight missing functionalities.

Eighteen problem observations were task failures, but only seven were unique. Furthermore, the clinicians mentioned several missing functionalities and bugs, which indicates their expectations and the need for improvements. Figure 8 shows all of the problems recorded in this study.

Usability Problems

Below we comment on some of the observations.

Problem type	Clinician 1	Clinician 2	Observed	Problems
Task failure	8	10	18	7
Cumbersome	3	1	4	3
Minor	6	4	10	3
Medium	4	0	4	4
Bug	4	3	7	4
Missing Functionality	5	5	10	8
Total			53	29

Figure 8. Number of problem observations and number of distinct usability problems for the second usability study. Most of the distinct problems were observed several times.

(1) Clinicians had difficulties binding controls to data. Both of them were unable to write the complex formula that joins three tables and used a Group By. However, from their comments we noticed that they knew what should be shown, but were not sure how to express it with a formula. Moreover, the Group By increased the level of complexity, and both clinicians asked for a simple way to specify a Group By. At the beginning of the study they were skeptical about the syntax, but as they progressed it became easier.

(2) Although the first clinician had some prior knowledge of formulas, we observed that in several cases he was confused by the *bang* and *dot* operators. On the other hand, he was able to bind correctly to data. Both of them could guess what the join-left -< operator meant by referring to the E/R Model, but using it in practice was difficult.

(3) Clinicians found the Design Panel and the WYBIWYS feature useful. We observed that getting immediate feedback helped them to reflect on formulas and adjust them properly. Auto-Completion was appreciated and used all of the time. At some point the first clinician commented "I noticed that Auto-Completion is not

suggesting anything. It is because I have to define the Rows property." This shows that the clinician was using it not only to avoid misspelling, but also to reflect on what he was doing.

(4) Several times they got information from the various panels. From our observations, this helped them to confirm what was visualized, view the changes from different perspectives, and understand the formula language better. We noticed that the first clinician used the Control-Data Hierarchy extensively, unlike the second clinician. The second clinician liked DataView and used it in combination with the E/R Model. The first clinician consulted the E/R Model panel, but DataView was not used.

Questionnaires

As in the first study, the questionnaires captured the subjective ratings about general aspects of uVis, the formula language, and features and panels of the development environment. Both clinicians rated the following features high: the Design Panel, the Modes, the E/R Model, and the Auto-Completion. uVis operators were difficult to use, especially for the second clinician. More drag-and-drop features were welcome.

Debriefing

The first clinician said "The uVis language requires training, and I have no suggestions how to improve it. A great thing of the Studio is the Design Panel placed in the center, where you can see the results as you work. The Auto-Completion tells you what you can write, this is great." The second clinician commented "I use databases because in MS Excel you cannot use SQL to access the database. This (uVis) is another way, but I can understand it. Also, I can understand the properties because I know them from Visual Basic. It is extremely important that we get some tools like this (uVis). I would prefer something that does not require any IT skills at all. I think there are some clinicians who can use tools like uVis. But it has to be more user-friendly and the database concepts are challenging as most clinicians are used to Spreadsheets and don't use databases."

DISCUSSION

Based on the studies, we have identified several issues that could improve end-user developers' ability to develop visualizations.

(1) The formula language (e.g., keywords and operators) was not as intuitive to participants as we expected. This problem is shared with spreadsheet users.

(2) According to Ref. 36, "direct manipulation systems offer the satisfying experience of operating on visible objects." Our solution provided several drag-and-drop mechanisms, but the users asked for more.

(3) The study in Ref. 37 showed that multiple coordinated views improve understandability and enable discovery of unanticipated relationships. We developed a coordinated environment, but it should be improved.

(4) Participants were enthusiastic about seeing the result of changes without having to switch environment. This should be kept.

(5) Data binding is one of the steps of the visualization development process.[14] It was the most difficult part observed in our studies. Novel solutions are needed.

(6) Noticing errors is difficult for our audience, but according to Ref. 15 it is not an issue with programmers. Indicating the errors in the formulas as well as in a list of errors is an improvement, but not sufficient.

Limitations

The studies have several limitations. In this work, we used usability studies. A different approach would be to run a controlled experiment where some participants used uVis and others used another tool to create the same visualization. However, a controlled experiment would not provide us with the necessary information to improve the tool.

The number of test subjects in the first study was only nine. This is inadequate for finding statistical significance, but according to Refs. 38, 39 it can provide a good indication of the existing problems. As more subjects are included, fewer and fewer new problems are observed. The second study was restricted to two participants, because recruiting clinicians is difficult. Most likely additional problems would be identified with more clinicians.

We have not tried to find out whether end-user developers can invent useful visualizations. This is a completely different question. What we want to know is whether they can implement the visualizations they might invent. We believe that innovation is related to the ability to experiment with visualizations, and here uVis seems to be a good possibility.

We are aware that some visualizations (e.g., network visualization) cannot currently be carried out with uVis, but this is a different matter from ease of use.

Since the uVis formulas are somewhat similar to spreadsheet formulas, we might have compared the usability of spreadsheets with the usability of uVis. We have not done this since it seems irrelevant for further development of uVis. We know, for instance, that end-user developers find binding to the database a hard part of uVis. Here, there is no inspiration from spreadsheets. Spreadsheet formulas cannot refer directly to the database. The query has to be imported to the spreadsheet cells.

CONCLUSION

uVis is a visualization tool for end-user developers. We have investigated to what extent uVis actually can be used by them. We performed two usability studies, one with a mix of potential end-user developers and one with two IT-interested clinicians. Participants were asked to create visualizations with uVis and evaluate the tool.

The results showed that with a modest amount of training, end-user developers can construct visualizations with uVis. They appreciated the simple formula language, the coordinated panels, and the drag-and-drop mechanism.

However, the most important thing for them was the immediate response when they changed something, for instance part of a formula. The entire visualization was updated immediately without having to switch from development view to production view.

The results of the studies also showed that the main difficulties for end-user developers are data binding and error noticing.

REFERENCES

[1] J. J. Thomas and K. A. Cook, "A visual analytics agenda," IEEE Comput. Graph. Appl. **26**, 10–13 (2006).

[2] L. C. Koh, A. Slingsby, J. Dykes, and T. S. Kam, "Developing and applying a user-centered model for the design and implementation of information visualization tools," *15th Int'l. Conf. on Information Visualisation* (2011), pp. 90–95.

[3] J. Pane and B. Myers, "More natural programming languages and environments," *End User Development*, edited by H. Lieberman, F. Patern and V. Wulf, Human–Computer Interaction Series (Springer, Netherlands, 2006), Vol. 9, pp. 31–50.

[4] H. Lieberman, F. Patern, M. Klann, and V. Wulf, "End-user development: an emerging paradigm," *End User Development*, edited by H. Lieberman, F. Patern and V. Wulf, Human–Computer Interaction Series (Springer, Netherlands, 2006), Vol. 9, pp. 1–8.

[5] H. Lieberman, F. Paternò, and V. Wulf, *End User Development (Human–Computer Interaction Series)* (Springer-Verlag, New York, Inc., Secaucus, NJ, USA, 2006).

[6] C. Scaffidi, M. Shaw, and B. Myers, "Estimating the numbers of end users and end user programmers," *IEEE Symposium on Visual Languages and Human-Centric Computing* (IEEE, Piscataway, NJ, 2005), pp. 207–214.

[7] W. Aigner, S. Miksch, W. Müller, H. Schumann, and C. Tominsky, "Visual methods for analyzing time-oriented data," IEEE Trans. Vis. Comput. Graphics **14**, 47–60 (2008).

[8] J. Heer, F. Ham, S. Carpendale, C. Weaver, and P. Isenberg, "Creation and collaboration: engaging new audiences for information visualization," *Information Visualization*, edited by A. Kerren, J. Stasko, J. D. Fekete and C. North, Lecture Notes in Computer Science (Springer, Berlin, Heidelberg, 2008), Vol. 4950, pp. 92–133.

[9] C. Plaisant, "The challenge of information visualization evaluation," *Proc. Working Conf. on Advanced Visual Interfaces. AVI '04* (ACM, 2004), pp. 109–116.

[10] M. Klann, F. Patern, and V. Wulf, "Future perspectives in end-user development," *End User Development*, edited by H. Lieberman, F. Patern and V. Wulf, Human–Computer Interaction Series (Springer, Netherlands, 2006), Vol. 9, pp. 475–486.

[11] J. Rode, M. Rosson, and M. Quiñones, "End user development of web applications," *End User Development*, edited by H. Lieberman, F. Patern and V. Wulf, Human–Computer Interaction Series (Springer, Netherlands, 2006), Vol. 9, pp. 161–182.

[12] K. Pantazos, S. Lauesen, and R. Vatrapu, "End-user development of information visualization," *IS-EUD*, edited by Y. Dittrich, M. M. Burnett, A. I. Mørch and D. F. Redmiles, Lecture Notes in Computer Science (Springer, 2013), Vol. 7897, pp. 104–119.

[13] A. MacLean, K. Carter, L. Lövstrand, and T. Moran, "User-tailorable systems: pressing the issues with buttons," *Proc. SIGCHI Conf. on Human Factors in Computing Systems. CHI '90, New York, NY, USA* (ACM, 1990), pp. 175–182.

[14] *Readings in information visualization: using vision to think*, edited by S. K. Card, J. D. Mackinlay, and B. Shneiderman (Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1999).

[15] K. Pantazos, M. A. Kuhail, S. Lauesen, and S. Xu, "uVis Studio: an integrated development environment for visualization," Proc. SPIE **8654-21**, (2013).

[16] M. Bostock, V. Ogievetsky, and J. Heer, "D3 data-driven documents," IEEE Trans. Vis. Comput. Graphics **17**, 2301–2309 (2011).

[17] M. Bostock and J. Heer, "Protovis: a graphical toolkit for visualization," IEEE Trans. Vis. Comput. Graphics **15**, 1121–1128 (2009).

[18] J. D. Fekete, "The infovis toolkit," *Proc. IEEE Symp. on Information Visualization* (IEEE, Piscataway, NJ, 2004), pp. 167–174.

[19] Flare: Flare. http://flare.prefuse.org/, Accessed August, 2014.

[20] Excel, M. http://office.microsoft.com/en-us/excel/, Accessed August, 2014.

[21] Omniscope. http://www.visokio.com/, Accessed August, 2014.

[22] Spotfire. http://spotfire.tibco.com/, Accessed August, 2014.

[23] Tableau. http://www.tableausoftware.com/, Accessed August, 2014.

[24] J. Heer, S. K. Card, and J. A. Landay, "Prefuse: a toolkit for interactive information visualization," *Proc. SIGCHI Conf. on Human Factors in Computing Systems. CHI '05* (ACM, 2005), pp. 421–430.

[25] T. Isenberg, P. Isenberg, J. Chen, M. Sedlmair, and T. Moller, "A systematic review on the practice of evaluating visualization," IEEE Trans. Vis. Comput. Graphics **19**, 2818–2827 (2013).

[26] C. Plaisant, R. Mushlin, A. Snyder, J. Li, D. Heller, B. Shneiderman, and K. P. Colorado, "Lifelines: using visualization to enhance navigation and analysis of patient records," *Proc. 1998 American Medical Informatic Assn. Annual Fall Symposium* (1998), pp. 76–80.

[27] T. A. Slocum, D. C. Cliburn, J. J. Feddema, and J. R. Miller, "Evaluating the usability of a tool for visualizing the uncertainty of the future global water balance," Cartography Geographic Information Sci. 299–317 (2003).

[28] A. C. Robinson, J. Chen, E. J. Lengerich, H. G. Meyer, and A. M. MacEachren, "Combining usability techniques to design geo-visualization tools for epidemiology," Cartography Geographic Information Sci. **32**, 243–255 (2005).

[29] R. E. Roth, K. S. Ross, B. G. Finch, W. Luo, and A. M. MacEachren, A user-centered approach for designing and developing spatiotemporal crime analysis tools. geovistapsuedu (Norman 1988) (2009).

[30] D. A. Norman, *The Design of Everyday Things* (DoubledayBusiness, 1990).

[31] J. Nielsen, *Usability Engineering* (Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1993).

[32] P. P.-S. Chen, "The entity-relationship model toward a unified view of data," ACM Trans. Database Syst. **1**, 9–36 (1976).

[33] S. L. Tanimoto, "Viva: a visual language for image processing," J. Vis. Lang. Comput. **1**, 127–139 (1990).

[34] K. Pantazos, S. Tarkan, C. Plaisant, and B. Shneiderman, "Promoting timely completion of multi-step processes—a visual approach to retrospective analysis," Technical Report HCIL-2012-27, University of Maryland, Human Computer Interaction Lab (2012).

[35] S. Lauesen, *User Interface Design: A Software Engineering Perspective* (Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2005).

[36] B. Shneiderman, "Direct manipulation: a step beyond programming languages," Computer **16**, 57–69 (1983).

[37] C. North and B. Shneiderman, "Snap-together visualization: can users construct and operate coordinated visualizations?" Int. J. Hum.–Comput. Stud. **53**, 715–739 (2000).

[38] J. Nielsen and T. K. Landauer, "A mathematical model of the finding of usability problems," *Proc. INTERACT '93 and CHI '93 Conf. on Human Factors in Computing Systems, CHI '93* (ACM, 1993), pp. 206–213.

[39] R. A. Virzi, "Refining the test phase of usability evaluation: how many subjects is enough?" Hum. Factors **34**, 457–468 (1992).